I'm not robot

Pdfsharp merge pdf with form fields

Merge pdf c#. Pdfsharp fill form. Pdf form mail merge. Pdfsharp merge pdf.

PDFs have a unique property: all fields with the same name share the same value. This is convenient when multiple forms refer to the same person, but it becomes an issue when trying to merge multiple instances of the same form into one PDF. One common solution is to flatten the forms before merging them, which effectively renders the merging process unnecessary. Another option is renaming fields prior to merging, but this can lead to data extraction difficulties and script breakages. Due to its complexities, flattening has become a more popular choice. iText's PdfCopyFields class allows for correct field copying between documents, including merging fields with the same name. However, it does not rename fields automatically; instead, it requires manually renaming each field prior to merging using the AcroFields object and its renameField method. For XFA forms (LiveCycle Designer-generated forms), dealing with XML becomes a significant task, especially when combining forms from different sources. Assuming one is working with AcroForm fields, code for merging multiple forms into one can look like this: ```java public void mergeForms(String outpath, String inPaths[]) { PdfCopyFields copier = new PdfCopyFields(new FileOutputStream(outpath)); for (String curInPath : inPaths) { PdfReader reader = new PdfReader(curInPath); renameFields(reader.getAcroFields()); copier.addDocument(reader); } copier.close(); } private static int counter = 0; private void renameFields(AcroFields fields) { Set fieldNames = fields.getFields().keySet(); String prepend = String.format("_%d.", counter++); for (String fieldName : fieldNames) { fields.rename(fieldName, prepend + fieldName); } } ``` Ideally, the `renameFields` method would also create a generic parent field and make all other fields in the document its children to enhance compatibility with Acrobat/Reader. This approach can help avoid unnecessary "save changes?" requests when closing the resulting PDF. The Syncfusion .NET PDF library provides comprehensive support for managing form fields within PDF documents, including creating, customizing, and implementing various use cases of PDF forms. Some key aspects covered in this repository include exploring different PDF form implementations using the Syncfusion .NET PDF library. With Syncfusion's .NET PDF library, users can create and customize fillable PDF forms with various form fields. These fields include text boxes, combo boxes, radio buttons, list boxes, check boxes, signature fields, and button fields. To start, a new PDF document is created using the `PdfDocument` class. A page is then added to the document using the `PdfPage` class. The standard font for the page can be set using the `PdfStandardFont` class. A text box field can be created for each form field using the `PdfTextBoxField` class. This field can be set to a specific name and have its bounds defined using the `RectangleF` class. A tooltip can also be added to provide additional context. The form fields are then added to the PDF document's form using the `Form.Fields.Add()` method. In this example, text box fields are created for "Name", "Email address", and "Phone". Radio button fields are used to create a field for "Gender" with options for "Male" and "Female". Finally, the PDF document is saved to a file stream using the `FileStream` class. To create, fill, and modify PDF documents using C#, you can utilize the PdfLoadedDocument class. Here's an example of how to accomplish this: ```csharp //Load the PDF document PdfLoadedDocument loadedDocument = new PdfLoadedDocument(docStream); //Load the form from the loaded document. PdfLoadedForm form = loadedDocument.Form; //Load the form field collections from the form. PdfLoadedFormFieldCollection fieldCollection = form.Fields as PdfLoadedFormFieldCollection; //Get and fill the fields using TryGetField Method if (fieldCollection.TryGetField("Name", out PdfLoadedField loadedField)) { (loadedField as PdfLoadedTextBoxField).Text = "Simons"; } if (fieldCollection.TryGetField("Email address", out loadedField)) { (loadedField as PdfLoadedTextBoxField).Text = "simonsbistro@outlook.com"; } if (fieldCollection.TryGetField("Phone", out loadedField)) { (loadedField as PdfLoadedTextBoxField).Text = "31 12 34 56"; } if (fieldCollection.TryGetField("Gender", out loadedField)) { (loadedField as PdfLoadedRadioButtonListField).SelectedIndex = 0; } ``` To modify and save a PDF, first load it as a document. Then access its form fields and delete any unwanted ones. For instance, you can remove text fields by loading them into a field object and then using the `Remove` or `RemoveAt` methods. The following code demonstrates how to do this: ```csharp // Load the PDF document. PdfLoadedDocument loadedDocument = new PdfLoadedDocument(docStream); // Get the loaded form. PdfLoadedForm loadedForm = loadedDocument.Form; // Load the text box field. PdfLoadedTextBoxField loadedTextBoxField = loadedForm.Fields[0] as PdfLoadedTextBoxField; // Remove the field. loadedForm.Fields.Remove(loadedTextBoxField); // Remove the field at index 0. loadedForm.Fields.RemoveAt(0); ``` Creating a signature field in a new PDF document involves adding a page to the document, creating a signature field on that page, and setting properties like bounds and tooltips. The following code example shows how to do this: ```csharp // Create a new PDF document. PdfDocument document = new PdfDocument(); // Add a new page to PDF document. PdfPage page = document.Pages.Add(); // Create PDF Signature field. PdfSignatureField signatureField = new PdfSignatureField(page, "Signature"); // Set properties to the signature field. signatureField.Bounds = new RectangleF(0, 100, 90, 20); signatureField.ToolTip = "Signature"; // Add the form field to the document. document.Form.Fields.Add(signatureField); // Save the PDF document. using (FileStream outputFileStream = new FileStream("Output.pdf", FileMode.Create, FileAccess.ReadWrite)) { document.Save(outputFileStream); } // Close the document. document.Close(true); ``` To digitally sign and verify signatures in PDF files using C#, refer to this blog post: "Digitally Sign and Verify Signatures in PDF Files Using C#". Syncfusion .NET PDF library allows you to flatten a form field or entire form by removing existing fields and replacing them with graphical objects that cannot be edited. The following code example demonstrates how to do this: ```csharp // Load the PDF document. PdfLoadedDocument loadedDocument = new PdfLoadedDocument(docStream); // Get the loaded form. PdfLoadedForm loadedForm = loadedDocument.Form; PdfLoadedFormFieldCollection fields = loadedForm.Fields; // Flatten the whole form. loadedForm.Flatten = true; // Save the PDF document. using (FileStream outputFileStream = new FileStream("Output.pdf", FileMode.Create, FileAccess.ReadWrite)) { loadedDocument.Save(outputFileStream); } // Close the document. loadedDocument.Close(true); ``` Finally, to set a PDF document as read-only, you can use the following code: ```csharp // Load the PDF document. PdfLoadedDocument loadedDocument = new PdfLoadedDocument(docStream); // Set the form as read-only. loadedForm.Flatten = true; ``` The code snippet provided demonstrates how to work with PDF forms using the Syncfusion .NET PDF library. It covers importing form fields data from an FDF file, exporting data in various formats such as FDF, XFDF, XML, and JSON, and preserving extended rights in PDF forms. To import form fields data, you can use the `ImportDataFDF` method, which takes a file stream and a boolean value indicating whether to overwrite any existing fields. The code example shows how to load an existing form and then import the FDF file into it. For exporting data, you can use the `ExportData` method, which allows you to specify the format of the exported data using the `DataFormat` enum. The example demonstrates how to export data in FDF format, specifying the name of the form as "AcroForm1". Finally, the code snippet highlights how to preserve extended rights in PDF forms when filling out form fields. This feature is particularly useful for users who require more advanced capabilities beyond the standard PDF specification. Here's a paraphrased version of the provided text: The Syncfusion .NET PDF library offers robust support for working with PDF forms, including importing and exporting form data. You can import FDF files into your PDF documents using `ImportDataFDF`, which allows overwriting existing fields. The library also enables you to export data in various formats such as XFDF, JSON, and XML through the `ExportData` method. Additionally, it provides the ability to preserve extended rights in PDF forms when filling out form fields. PdfLoadedFormFieldCollection fieldCollection = loadedForm.Fields as PdfLoadedFormFieldCollection; PdfLoadedField loadedField = null; //Get the field using TryGetField Method. if (fieldCollection.TryGetField("First Name", out loadedField)) { (loadedField as PdfLoadedTextBoxField).Text = "Simons"; } if (fieldCollection.TryGetField("Last Name", out loadedField)) { (loadedField as PdfLoadedTextBoxField).Text = "Bistro"; } if (fieldCollection.TryGetField("Email Address", out loadedField)) { (loadedField as PdfLoadedTextBoxField).Text = "simonsbistro@outlook.com"; } using (FileStream outputFileStream = new FileStream("Output.pdf", FileMode.Create, FileAccess.ReadWrite)) { //Save the PDF document to file stream. loadedDocument.Save(outputFileStream); } //Close the document. loadedDocument.Close(true); PdfDocument document = new PdfDocument(); PdfPage page = document.Pages.Add(); PdfButtonField submitButton = new PdfButtonField(page, "submitButton"); submitButton.Bounds = new RectangleF(25, 160, 100, 20); submitButton.Text = "Apply"; submitButton.BackColor = new PdfColor(181, 191, 203); PdfJavaScriptAction scriptAction = new PdfJavaScriptAction("app.alert(\"You are looking at form fields action of PDF document\")"); submitButton.Actions.MouseDown = scriptAction; document.Form.Fields.Add(submitButton); using (FileStream outputFileStream = new FileStream("Output.pdf", FileMode.Create, FileAccess.ReadWrite)) { //Save the PDF document to file stream. document.Save(outputFileStream); } document.Close(true); We offer a wide range of components and frameworks for web, mobile, and desktop development, including Blazor, ASP.NET Core, ASP.NET MVC, and more. Additionally, our software is ready-to-deploy for dashboards, reports, data integration, and big data processing, having helped customers save millions in licensing fees.